CS489 Report: Flute Synthesis

Dave Pagurek (dpagurek, 20551040)

April 2019

Contents

Motivation

I record music as a hobby, and the music I produce occasionally includes recordings of my own flute playing. I do this because I can get a level of expressivity on the flute that I can't achieve as easily with the synthesizers I work with. I set out to construct a synthesizer that lets me imitate that sound, including the ability to control the intensity, vibrato, breathiness, and other parameters that I make use of when playing a real instrument. I hoped to create a tool that would give me more flexibility when prototyping new pieces, both in terms of the ease with which I can tweak and experiment and also the freedom of not having to disturb roommates should I decide to work at inconvenient times of the day.

My goal was both to produce a flute synthesizer, but also to gain a better understanding of the underlying processes making the flute sound like a flute. So, to begin with, I turned to theory and data to model the harmonics present in the sound of the flute.

Flute harmonics

The theory

The flute can be modelled as an open air column [1] for the purposes of examining its resonance. Resonant frequencies arise in open air columns due to the standing pressure wave patterns that are able to form. The geometry of the column allows standing waves with integer number of nodes, shown in Figure 1.

Each standing wave has a wavelength relative to half the length of the instrument [2]. The frequency of the note produced by each wave is found by the equation $f = \frac{v}{\lambda}$, where v is the speed of sound, and λ is the wavelength. This tells us that the lowest resonant frequency is

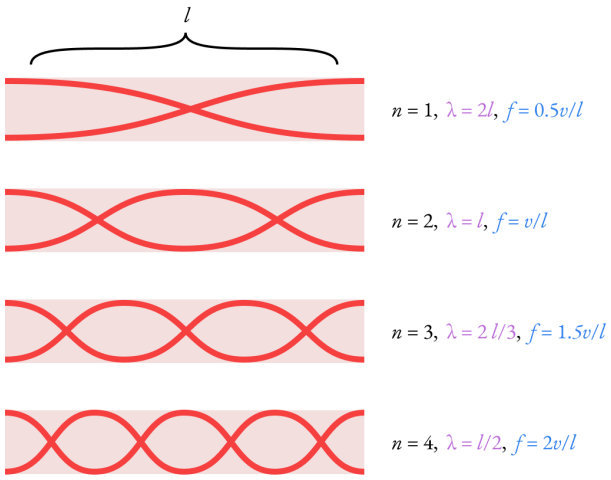

Figure 1: The frequency of a harmonic relative to the number of nodes. Each successive harmonic has a frequency which is an integer multiple of the lowest note that can be sounded.

proportional to twice the length of the instrument. Each successively higher frequency increases by a factor of half the length of the instrument, and every harmonic above that is an integer multiple of this lowest frequency.

Given that the length of a flute is around 66 cm, we can estimate the lowest note, corresponding to one node:

$$\begin{aligned} f &= \frac{v}{\lambda} \\ &= \frac{340.27 \text{ m/s}}{2 \cdot 66 \text{ cm}} \quad = 257.78 \text{ Hz} \end{aligned}$$

This frequency corresponds to the note B_3 and 74 cents. This agrees with the experience of playing the flute, where the lowest note that can be fingered is C_4 by covering every hole. If we continue adding doubling the frequency, we start filling in all the possible harmonics. Table 1 shows what notes these correspond to.[1]

Harmonic	**Frequency**	**Note**
1	261.6256	C_4
2	523.2511	C_5
3	784.8767	G_5 and 2 cents
4	1046.5023	C_6
5	1308.1278	$D_6^\sharp$ and 86 cents
6	1569.7534	G_6 and 2 cents
7	1831.3790	A_6 and 69 cents
8	2093.0045	C_7
9	2354.6301	D_7 and 4 cents
10	2616.2557	$D_7^\sharp$ and 86 cents

Table 1: Harmonics of C_4

[1]It is interesting to note here that the harmonic frequencies are not in tune. We perceive a doubling of a frequency to be a jump up an octave, so octaves are found to be proportional to 2^n for increasing n. Equal-tempered tuning divides the space between octaves into 12 equally spaced semitones, so semitones are proportional to $2^{\frac{n}{12}}$. Integer multiples of a base note do not always align with these twelfths. Some instruments purposefully detune notes that are intended to be played in a chord with other notes so that the harmonics do not interfere dissonantly with the rest of the chord [3].)

These are the frequency ratios that will have to be simulated in order to create a convincing flute synthesizer. Higher notes are made by uncovering holes on the flute to effectively shorten its length and increase the frequency of the lowest note that can be sounded.

The next piece of information required is the amplitude of each harmonic. This is what will make a flute sound like a flute. For that, I turn to data from actual recordings.

Harmonics in the wild

In 2018, I recorded a song [4] that featured recordings of myself playing the flute. As a starting point, I extracted four samples from one flute track in the song where a single note is held for around a second. There are samples for the notes G, F, E♭, and D. Each file is a mono wav file at a 44100 Hz, cut down to just be the steady-state portion of the held note. I processed each sample to get a sense of the harmonics present in the sound of the flute.

To do this, I ran each sample through a Fast Fourier Transform (FFT). The FFT values are normalized by multiplying the complex magnitude by $\frac{2}{N}$ corresponding to the N windowed values with a multiplier of 1. The values were then converted to decibel units. This converts each sample into the frequency domain, showing, for each frequency contributing to the overall sound, the gain on its contribution. The frequency domain is slightly noisy due in part to the background noise in the recording and the imperfections in my own playing, so frequency peaks tend to be spread out over a few frequency bands. I ran the FFT results through a peak finding algorithm to automatically identify, approximately, the peaks in the spectrum. The results are shown in Figure 2.

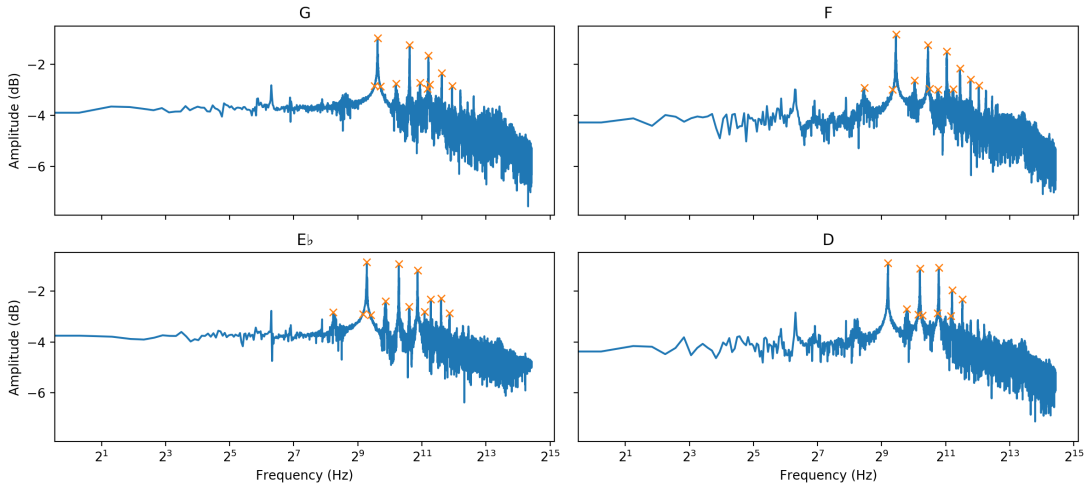

Figure 2: Frequency analysis of four notes. The frequency curves were run through a peak finding algorithm, and the identified peaks have been marked with an X.

For each peak found, I graphed its amplitude relative to the ratio of its frequency to the frequency of the note I was playing, shown in Figure 3. We can see that the peaks come in multiples of 0.5. Why does this occur?

All of these frequencies are integer multiples of the fundamental. If the harmonics look like they are all multiples of 0.5, like what we saw in the harmonics from the recordings, the actual fundamental frequency is half of the frequency of the note I was playing. Notes like this are played by overblowing to change the air pressure in the instrument, preventing the first harmonic from sounding.

To figure out which notes this applies to, I played an ascending scale and measured the ratios of the harmonics that sounded to the amplitude of the frequency I was attempting to play. The

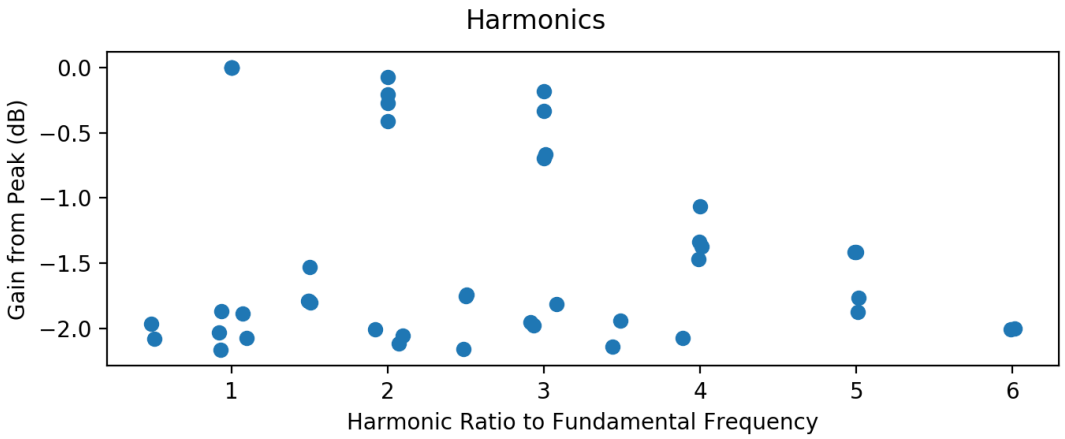

Figure 3: Distribution of the amplitude of frequency peaks found in recordings of four notes relative to the frequency and amplitude of the fundamental note being played.

results are shown in Figure 4. Notes for which harmonics come in multiples of 0.5 instead of 1 indicate overblown notes.

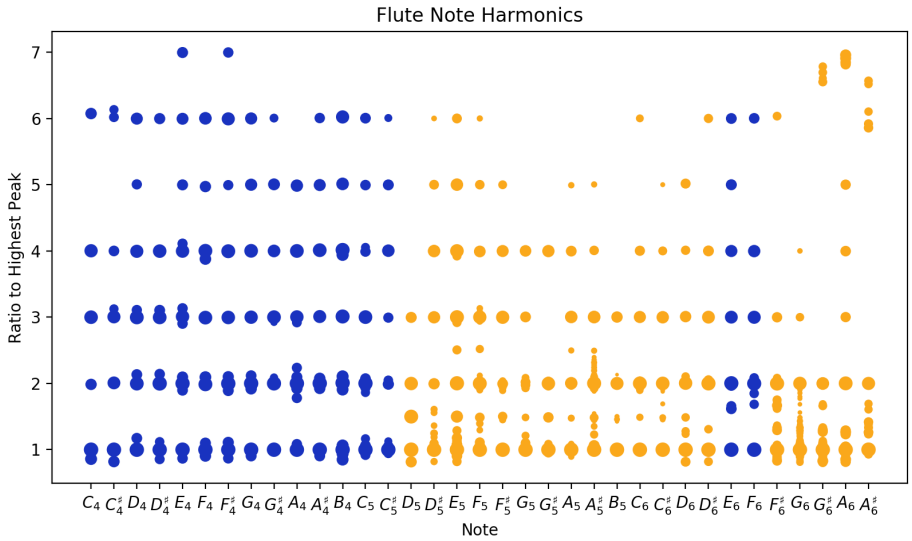

Figure 4: A chart showing a dot for each harmonic picked up by the peak finder on the FFT output, with dot size corresponding to relative amplitude. The overblown notes have been highlighted orange.

Given this information, I remade the earlier plot of harmonics and their amplitudes using all the scale samples. This time, I made two separate graphs, showing the different falloff for regular and overblown notes, shown in Figure 5.

The large spread of low amplitude values is largely due to noise surrounding the real peak that got picked up by the peak detector. Even in the clear peak regions, there is large variance. I've chosen to average the values into a plausible common amplitude that will sound close enough. The amplitudes I picked for each harmonic in additive synthesis are shown in Table 2.

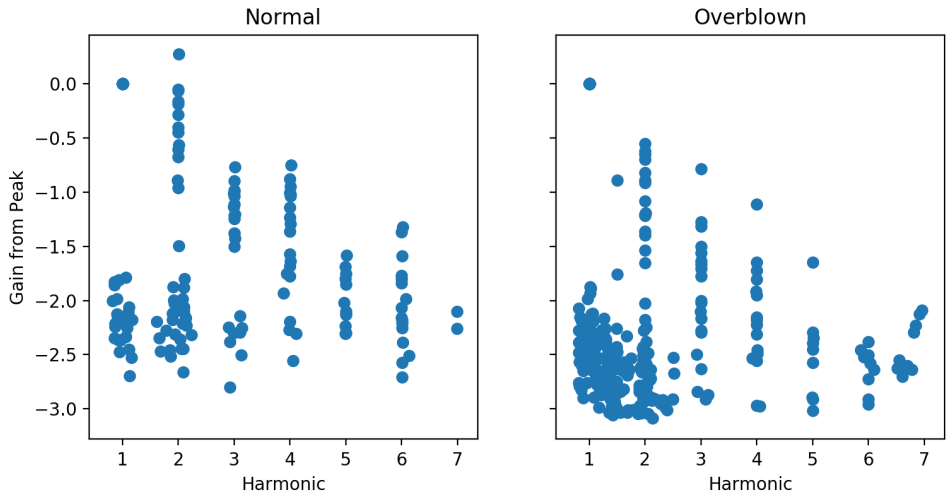

Figure 5: A plot of the relative amplitude of harmonics, segmented into the overblown and normal samples.

Harmonic	**Normal gain**	**Overblown gain**
1	0	0
1.5	-∞	-1.75
2	-0.5	-1
2.5	-∞	-2.5
3	-1.25	-1.75
4	-1.5	-2
5	-2	-2.5
6	-2.25	-2.6
7	-2.3	-2.7

Table 2: Harmonic amplitudes

Note envelope

The next question is how the amplitudes for each frequency change over time. Examining the amplitude for the entire waveform, a standard linear attack-decay-sustain-release function can fit decently. Figure 6 shows an ADSR envelope drawn on top of the waveform of a note played on the flute.

Do all the harmonics sound at the same time? To answer this question, I looked at short time Fourier transform graph of the same flute recording, shown in Figure 7. This is an overblown note, and the first approximately 0.02 seconds of the recording have a much larger contribution from the fundamental, non-overblown note, before the air rate stabilizes and the note settles on the second harmonic.

Figure 7 also demonstrates the fluctuation in amplitude in the harmonics of a held note. Contrast this with intentional vibrato, shown in Figure 8. There are two separate processes going on that shape the amplitude of harmonics: subtle random fluctuation from unconscious air rate changes, and intentional periodic modulation of just the upper harmonics.

I model the former by multiplying the ADSR envelope with Perlin noise [5]. This generates noise as a continuous function, with each "octave" of noise adding lower-amplitude, higher-frequency detail. Each harmonic oscillator gets its own two-octave noise track, starting at a random offset so that they do not modulate uniformly. The time in seconds is scaled by a factor of 10 before being sent into the noise function so that the amplitude changes quickly enough to be noticeable but not dramatic. I make the noise amplitude range from 0.6 and 1, effectively lowering the

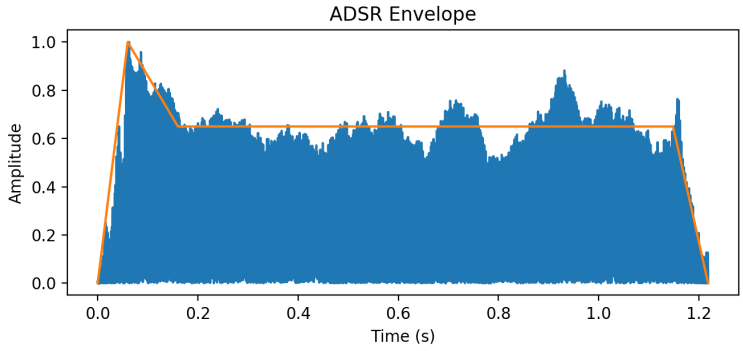

Figure 6: An attack time of 0.06s, a decay time of 0.1s, a sustained level of 0.65 times the peak, and a release time of 0.07s, shown on top of a held flute note.

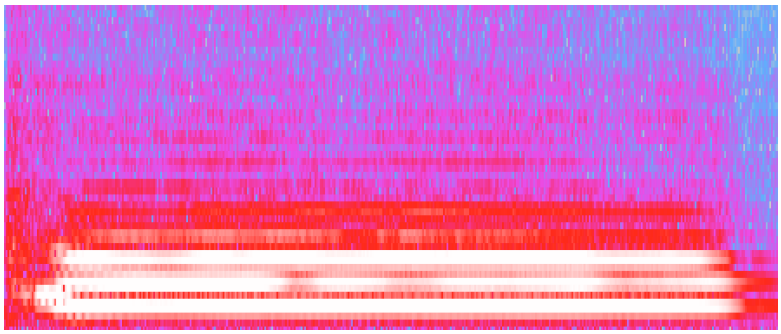

Figure 7: The short-time Fourier transform of D_4, showing how there is a delay before the overblown frequencies are sounded significantly.

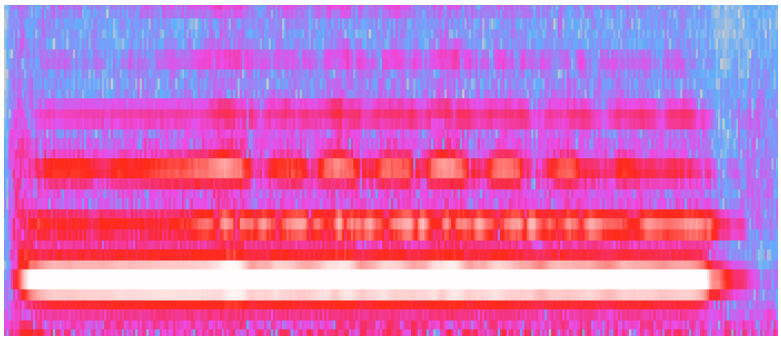

Figure 8: Vibrato on an overblown $B_5^\flat$.

amplitude randomly.

It is interesting that the "vibrato" here is not technically vibrato in the traditional sense. Vibrato is defined as a modulation in pitch, but that is not what we see here. It makes sense that we would not see a change in pitch, since flute vibrato is done entirely with air flow, which does not change the length of the instrument and therefore does not bend the pitch. It also is not traditional tremolo, although this gets close. Tremolo is amplitude modulation, which is what we see, except that it only affects the upper harmonics. Upper harmonic tremolo is perhaps more accurately defined as *brightness* modulation since it primarily affects the timber of the sound.

I model the brightness modulation with a 5Hz sine wave that gets multiplied with the note envelope. It cycles between amplitudes of 0.1 and 1.3, allowing the modulation to nearly completely suppress the harmonic at its low point and slightly boost it at its peak. When playing a note, the vibrato usually does not start immediately, perhaps not starting at all for short notes. To account for this, the sine wave is multiplied with a sigmoid function offset by 0.4s, with time scaled by a factor of 30. This allows the note to remain unmodulated for a moment before the sine wave fades in. This is applied only to harmonics greater than 2. The envelopes for both fluctuation and brightness modulation are shown in Figure 9 to compare their amplitudes and regularity.

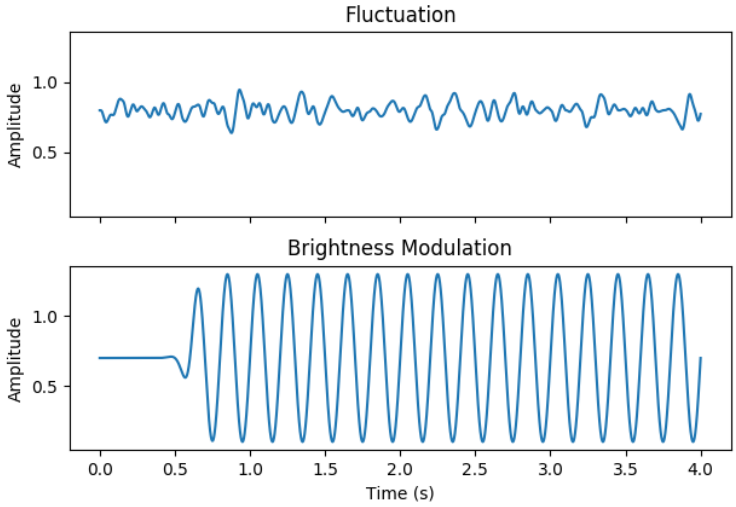

Figure 9: Fluctuation and brightness modulation ("vibrato") envelopes.

Noise component

A spectrum analysis of recorded notes shows that there is a noise component to the sound. Most of this low-energy noise comes from the air flow leaving the mouth, with some frequencies resonating in the tube of the instrument.

I chose to model this as white noise that gets filtered with resonance. White noise is defined by having equal contributions from each frequency, but there are multiple functions that generate discrete time-domain samples that create white noise. One could implement *Gaussian white noise*, where samples have a normal distribution with zero mean. However, Gaussian distributions have an unbounded range, and we need amplitudes to stay between -1 and 1. Instead, I used a uniform distribution to generate time-domain samples.

The noise then needs to be filtered down. A simple first-order system could be used to create a lowpass filter, but the rolloff rate after the filter cutoff is fixed at -20dB per decade, and it would

not model resonance. A second-order filter, on the other hand, allows for a steeper -40dB per decade rolloff and resonance. For that reason, I use a second-order system.

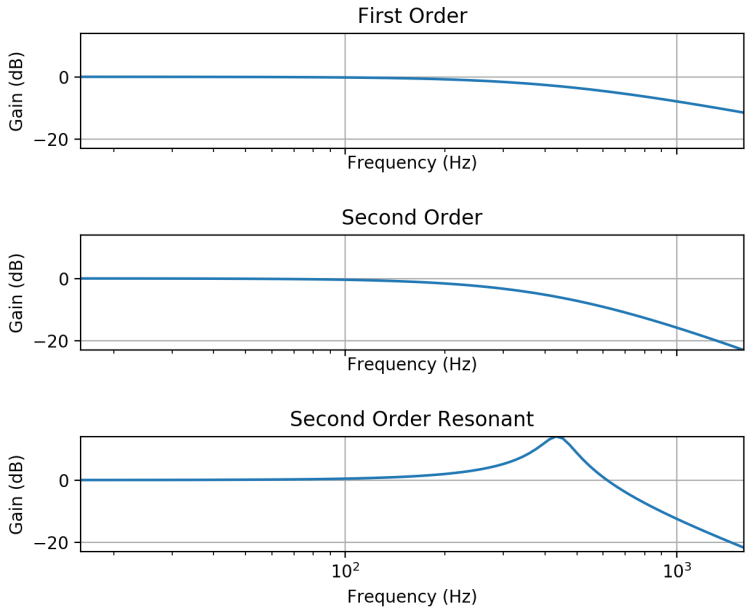

Figure 10: Bode plots showing the frequency response of first and second order low pass filters. The second order filter has a steeper rolloff.

Transfer functions for second order systems typically have the form $H(s) = \frac{K\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$. $\omega_n \approx \omega_{BW}$, which is the cutoff frequency after which we see rolloff. I set this equal to the frequency being sounded by the flute. No extra gain is needed, so I set the gain to $K = 1$. ζ dictates the oscillatory behaviour of the system: if it is greater than or equal to 1, the system is perfectly damped or overdamped, so it will not overshoot when following an input signal. If it is less than 1, the filter follows the input signal faster, but it will overshoot and oscillate when reaching to "catch up" with an input signal. This effect causes resonance, so I set $\zeta = 0.01$.

To actually implement this as a digital filter, the transfer function needs to be turned into a discrete time equation relative to the past discrete samples. The first step is to separate the frequency-domain input $X(s)$ and output $Y(s)$.

$$
\begin{aligned}
H(s) &= \frac{K\omega_{BW}^2}{s^2 + 2\zeta\omega_{BW}s + \omega_{BW}^2} \\
\frac{Y(s)}{X(s)} &= \frac{1}{\frac{s^2}{K\omega_{BW}^2} + \frac{2\zeta s}{K\omega_{BW}} + \frac{1}{K}} \\
X(s) &= Y(s)\left(\frac{s^2}{K\omega_{BW}^2} + \frac{2\zeta s}{K\omega_{BW}} + \frac{1}{K}\right) \\
X(s) &= \frac{s^2}{K\omega_{BW}^2}Y(s) + \frac{2\zeta s}{K\omega_{BW}}Y(s) + \frac{1}{K}Y(s)
\end{aligned}
$$

Having separated input from output, the equation can be brought into the time domain using the inverse Laplace transform, assuming $y(0) = \left.\frac{dy(t)}{dt}\right|_{t=0} = \left.\frac{d^2y(t)}{dt^2}\right|_{t=0} = 0$:

$$\mathcal{L}^{-1}\{X(s)\} = \mathcal{L}^{-1}\left\{\frac{s^2}{K\omega_{BW}^2}Y(s) + \frac{2\zeta s}{K\omega_{BW}}Y(s) + \frac{1}{K}Y(s)\right\}$$
$$x(t) = \frac{1}{K\omega_{BW}^2}\left(\frac{d^2y(t)}{dt^2}\right) + \frac{2\zeta}{K\omega_{BW}}\left(\frac{dy(t)}{dt}\right) + \frac{1}{K}y(t)$$

Finally, the first and second derivatives can be approximated using the discrete time samples $y(t)$, $y(t-\Delta t)$, and $y(t-2\Delta t)$, rearranging for the next output, $y(t)$:

$$x(t) = \frac{1}{K\omega_{BW}^2}\left(\frac{y(t) - 2y(t-\Delta t) + y(t-2\Delta t)}{\Delta t^2}\right) + \frac{2\zeta}{K\omega_{BW}}\left(\frac{y(t) - y(t-\Delta t)}{\Delta t}\right) + \frac{1}{K}y(t)$$
$$y(t) = \frac{x(t) - \left(\frac{-2\zeta}{K\omega_{BW}\Delta t} - \frac{2}{K\omega_{BW}^2\Delta t^2}\right)y(t-\Delta t) - \left(\frac{1}{K\omega_{BW}^2\Delta t^2}\right)y(t-2\Delta t)}{\frac{1}{K} + \frac{2\zeta}{K\omega_{BW}\Delta t} + \frac{1}{K\omega_{BW}^2\Delta t^2}}$$
$$\begin{aligned} y(t) = &\left(\frac{K\omega_{BW}^2\Delta t^2}{\omega_{BW}^2\Delta t^2 + 2\zeta\omega_{BW}\Delta t + 1}\right)x(t) \\ &- \left(\frac{-2\zeta\omega_{BW}\Delta t - 2}{\omega_{BW}^2\Delta t^2 + 2\zeta\omega_{BW}\Delta t + 1}\right)y(t-\Delta t) \\ &- \left(\frac{1}{\omega_{BW}^2\Delta t^2 + 2\zeta\omega_{BW}\Delta t + 1}\right)y(t-2\Delta t) \end{aligned}$$

Each new value outputted by the filter is effectively a weighted average of the new input and the past two output values. The concrete values of the weights are found by using $\Delta t = 44100\text{Hz}$, the sampling frequency, and by setting ω_{BW}, ζ, and K to the aforementioned values. Figure 11 shows what white noise looks like, subjected to a first-order lowpass filter and two second-order filters.

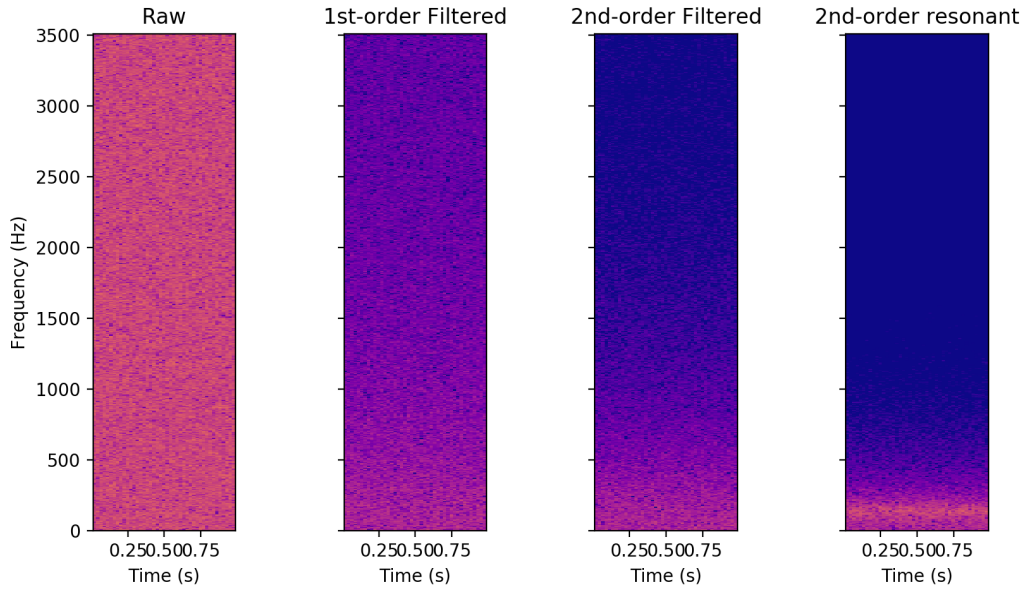

Figure 11: White noise, shown raw, and subjected to first- and second-order filters, with and without resonance.

MIDI integration

I wrote code to import a MIDI file as input, allowing composition to happen in a standard Digital Audio Workstation before rendering a final sound file using the synthesizer. MIDI notes come in the form of note-on and note-off events, indicating where each pitch begins sounding and stops being sounded. The note-on event additionally comes with a *velocity* parameter, indicating the force with which the note should be sounded. This is the main dial a composer can tweak when writing sequenced digital music. I implemented parameter variation based on the pitch and velocity to increase the realism of a full synthesized score:

- **Overall amplitude scales down as velocity scales down.** Velocity input, after normalization, comes as a number in $[0, 1]$. The overall magnitude of the synthesizer output is multiplied by the velocity.
- **Noise amplitude scales based on pitch.** Very low and very high notes require more breath control to play on a flute. Consequently, lower-register notes and higher-register notes tend to end up sounding noisier. To model this, the noise component is given a gain that increases from 0 dB to 1 dB as the pitch moves from an A at 440 Hz up or down an octave.
- **Higher notes get tuned slightly flat.** High notes require overblowing, which means an increased air speed in the flute column. This is achieved more easily when the head of the instrument is rolled in slightly, so that when air is split on the edge of the mouthpiece, more goes into the column. This also has the effect of lowering the pitch slightly.
- **Notes get initially overblown at high velocities.** Notes that are not already overblown are given a slight overblow effect at the beginning of a note, where there is an initial higher air speed. This is done by multiplying the peak volume for harmonics above the fundamental by a factor that increases from 1 to 1.2 as velocity increases from 0.8 to 1.

Results and limitations

A short-time Fourier transform graph of a real recording of a $B_5^\flat$ with vibrato is shown next to a graph of the synthesized version of the same thing in Figure 12. It is clear that there are higher harmonics missing in the synthesized version, but where each has increasingly lower frequency and where each becomes more "spread out," fading it into the background noise. Its omission in the synthesized version is noticeable, as the overall sound appears to be more muffled when compared with its real-life counterpart, but it still sounds convincing on its own.

I also took the score for a flute duet [6] I wrote last summer, generated a MIDI file, and produced a synthesized version. Audio files for the original and the synthesized version can listened to online [7] and compared. In general, the variations produced using pitch and velocity changes are effective at making the synthesized version sound dynamic and capable of an emotional delivery. However, due to the aforementioned differences in the upper harmonics, the synthesized version sounds noticeably less crisp. Additionally, the quick low notes in real recordings sound much richer and have much more variation than in the synthesized version. The increased noise component in the lower register helps, but a future iteration of this project would likely give more weight to harmonics for lower-register notes due to how much easier it is to overblow those notes.

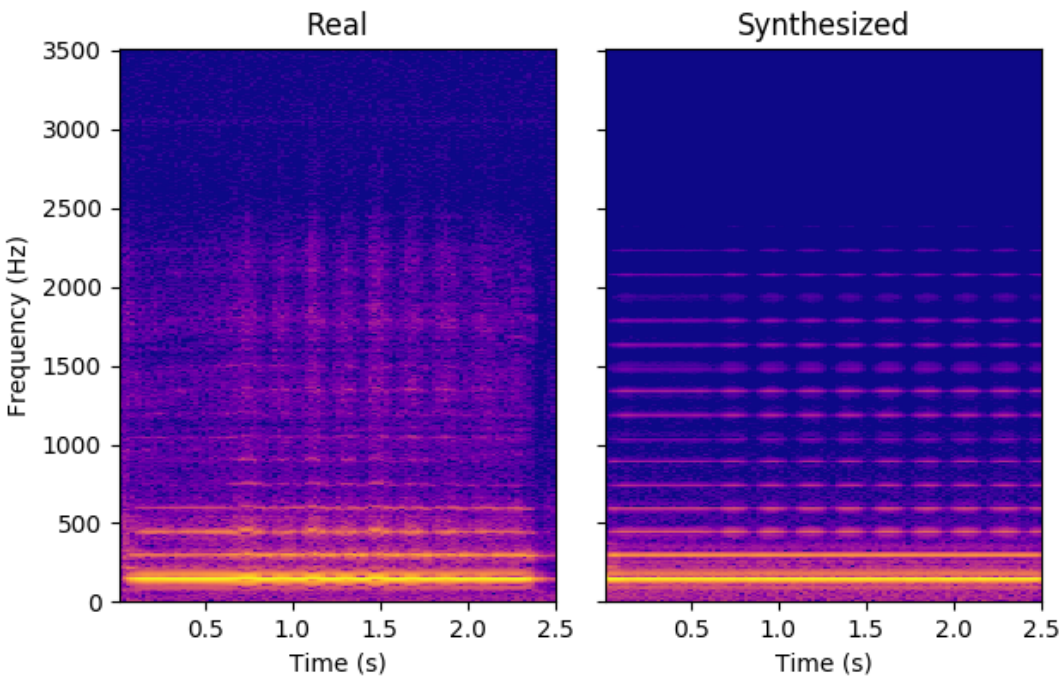

Figure 12: The frequencies over time for a real recording of a $B_5^\flat$ with vibrato compared with a synthesized version.

References

[1] C. R. Nave, “The flute” [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/Music/flute.html

[2] C. R. Nave, “Resonances of open air columns” [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/Waves/opecol.html

[3] Édouard, “Why are pianos traditionally tuned "out of tune" at the extremes?” 2015 [Online]. Available: https://music.stackexchange.com/questions/14244/why-are-pianos-traditionally-tuned-out-of-tune-at-the-extremes

[4] D. Pagurek, “Throw the dice.” 2018 [Online]. Available: https://soundcloud.com/davidpvm/throw-the-dice

[5] K. Perlin, “Improving noise,” in *Proceedings of the 29th annual conference on computer graphics and interactive techniques*, 2002, pp. 681–682 [Online]. Available: http://doi.acm.org/10.1145/566570.566636

[6] D. Pagurek, “Afar.” 2018 [Online]. Available: https://soundcloud.com/davidpvm/afar-arranged-for-flute-duet

[7] D. Pagurek, “Flute synthesis.” 2019 [Online]. Available: https://www.davepagurek.com/programming/flute-synthesis/